Chair of Software Engineering

Implementing and evaluating an exception mechanism for SCOOP

Project Plan

Florian Besser
ETH Zurich
fbesser@ethz.ch

March 18, 2013 - September 16, 2013

Supervised by:
Benjamin Morandi
Prof. Bertrand Meyer

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Contents

1 Project Description

1.1 Overview

SCOOP, a concurrency extension to the Eiffel language, was first conceived by Bertrand Meyer in the early nineties. There were many changes to SCOOP, and it finally became a part of EiffelStudio in 2011. There are still some problems when it comes to handling exceptions with SCOOP, though.
Exceptions are events requiring special processing, which often change the normal flow of how programs are executed. They generally are thrown if a feature encountered an error, and has to return unexpectedly. When an exception is thrown and not handled by the current feature, it is then passed onto the caller. In concurrent settings, such as SCOOP, this can be problematic, since a caller that receives the thrown exception might not be in any position to handle it.
The next logical step to improve SCOOP was then to develop a model for handling such situations, see [1]. The aim of this thesis is to implement that model, evaluate it thoroughly and report the findings.
To achieve this, some of the foundation of how the Eiffel compiler transforms SCOOP programs into C code has to be changed.

1.2 Intended Results

A complete exception mechanism for the SCOOP programming model according to the specifications will be developed.
An evaluation will then highlight the usability, expressiveness and performance of the implementation. This will be done using several small SCOOP applications, which will also be developed.
Improvements upon the original implementation will be constructed based on the evaluation.
(Optional) At the end, an implementation of the "Duel" mechanism, described in [3], page 999+, will be attempted.

1.3 Criteria for Success

Complete all objectives listed under work packages, see Section 3.1.

2 Background Material

2.1 Reading List

[1] describes the way the exception mechanism should perform, as well as some explanations why it should do so.
[2] describes the operational semantics of SCOOP.

3 Project Management

3.1 Components of the Solution

The solution consists of the following parts:

1. Implementation: The behavior specified in [1] will be implemented in the EVE environment. Programs written and compiled using EVE should then be able to handle exceptions as defined.
2. Evaluation programs: Several basic SCOOP programs relying on exceptions will be written. These programs serve to verify and evaluate the usability of the proposed mechanism.
3. Evaluation: The evaluation will focus on usability, expressiveness and performance of the implementation.
4. Improvements: The mechanism will be improved with respect to the findings of the evaluation. The system described in [1] might allow some improvements which make it more usable, more expressive or just plain faster.
5. Documentation:
   - A user guide on how to use the new functionality will be written. The user guide will include examples as well as explanations for best practises. It will enable a developer with basic knowledge of SCOOP to gain insights on how exceptions work.
   - The developer guide will contain precise information on the code and will allow future developers to easily change and improve the codebase. It will include explanations on how the system is designed and how a typical excecution might look. It will also include some reasoning as to why design decisions were made the way they were made, to give future developers the ability to understand and later improve the code, and not just change it.
6. (Optional) The "Duel" mechanism: The mechanism as described in [3], page 999+, will be implemented.

3.2 Project Steps / Milestones

These are the project milestones:

1. Reading parts of the codebase of EVE / SCOOP to get an overview of the system. After this step it should be possible to predict what code will be executed when presented with an example program.
2. Create basic test programs. Basic programs make use of SCOOP and throw exceptions, but do not use advanced features of the Eiffel language, such as once routines. They also do not rely on the yet to be implemented SCOOP "safe mode".

3. Implement basic system as specified in [1]. This should produce a system that can handle the basic programs created in milestone 2. Programs not relying on exceptions should still work just as they did before the change.

4. Extend system to handle advanced language components, such as once routines, the SCOOP "safe mode", exceptions from failed correctness conditions and separate callbacks. Write advanced programs, which possibly use all language features Eiffel offers to test the system. The system should be able to handle complex programs after this milestone.

5. Evaluate system, with respect to the programs written before, for usability and expressiveness. After evaluation, benchmark programs created for milestones 2 and 4 to gauge efficiency of the implementation.

6. Improve system based on the evaluation, then go back to step 5. After each improvement, the programs written for milestones 2 and 4 are executed again, to make sure no bugs were introduced. Clean up code where necessary, so that it is easily readable and maintainable for future developers.

7. (Optional) Research and implement the "Duel" mechanism described in [3], page 999+.

8. Write thesis and finish documentation. Write both the user guide as well as the developer guide.

3.3 Schedule

Milestone	
1	2 Weeks
2	2 Weeks
3	3 Weeks
4	5 Weeks
5	1 Week
6	1 Week
7	2 Weeks
8	3 Weeks
Total	$\approx$ 23 Weeks
	(Assume 3 Repetitions of Tasks 5 & 6)

3.4 Method of Work

- Bi-weekly meetings
- EVE for the implementation
- LaTeX for the documentation and thesis
- Iterative development
- Version control system provided by the EVE research group

3.5 Deadline

16. September 2013

4 References

[1] B. Morandi, S. Nanz, and B. Meyer. Who is Accountable for Asynchronous Exceptions?
[2] B. Morandi, S. Nanz, and B. Meyer. A comprehensive operational semantics of the SCOOP programming model. Arxiv preprint arXiv:1101.1038, 2011.
[3] B. Meyer, Object-Oriented Software Construction, Second Edition, Prentice Hall, 1997.